This excerpt is taken from the TK Solver 5.0 User Manual to give you a better idea of how the Solution Optimizer can help you accomplish more in less time. The solution Optimizer, included in the TK Solver 5.0 Premium Edition, is a valuable tool that gives you advanced control over constraints and bounds to help you achieve optimal design parameters—no programming required! You'll like the easy–to–use, Wizard–driven system that readily walks you through the process and empowers you to take your design to a whole new level! See what the Solution Optimizer can do for you today!

TK Solver 5.0 User's Guide

Universal Technical Systems, Inc.

Credits Page

Copyright

The UTS documentation and the software are copyrighted with all rights reserved. Under the copyright laws, neither the documentation nor the software may be copied, photocopied, reproduced, translated, or reduced to any electronic or machine readable form, in whole or part, without the prior written consent of UTS.

Copyright © 1983-2013 Universal Technical Systems, Inc.
Universal Technical Systems, Inc.
4053 North Perryville Road, Loves Park, Illinois 61111
Phone (815) 963-2220 Fax (815) 963-8884

All rights reserved. First edition printed in August 1994 in the U.S.A. Revised 2004.

Contains images by Corel Corp., 1600 Carling Ave., Ottawa, Ontario, Canada K1Z 8R7, (613) 728-3733.

Contains Macromedia Flash™ Player software by Macromedia, Inc., Copyright © 1995-1999 Macromedia, Inc. All rights reserved. Macromedia and Flash are trademarks of Macromedia, Inc.

Contains Macromedia Shockwave™ Player software by Macromedia, Inc., Copyright © 1995-1999 Macromedia, Inc. All rights reserved. Macromedia and Shockwave are trademarks of Macromedia, Inc."

Trademarks

Following is a list of Trademarks™ or Registered Trademarks® and the companies by which they are owned:

TK Solver for Windows, TK Solver, TK, TK Solver Plus, TK Plus, TK!Solver, Roark & Young Demo, MiniTK, Roark & Young on TK, Heat Transfer on TK, Dynamics & Vibration Analysis, Gear Analysis Software, DesignLink, Queueing Analyses, Tolerance Analysis, TK/CAD Link and MathLook by Universal Technical Systems, Inc.

Windows, Word, Excel, and Visual Basic by Microsoft Corporation.

Limitation of Liability

Universal Technical Systems, Inc. (UTS) does not warrant that the software program will function properly in every hardware/software environment. The software may not work in combination with modified versions of the operating system, with certain print-spooling or file facility programs or with certain printers supplied by independent manufacturers.

Although UTS has tested the software and reviewed the documentation, UTS makes no warranty or representation either expressed or implied, with respect to this software or documentation, including their quality, performance, merchantability, or fitness for a particular purpose. As a result, this software and documentation are licensed “as is” and you, the licensee, are assuming the entire risk as to their quality and performance.

In no event will UTS be held liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the software or documentation, even if advised of the possibility of such damages. In particular, UTS is not responsible for any costs including but not limited to those incurred as a result of lost profits or revenue, loss of use of the computer program, loss of data, the cost of substitute programs, claims by third parties or for other similar costs.

TK5-090904

Solution Optimizer 7

Solution Optimizer is included with the Premium version of TK Solver. If you do not have this feature and would like to order it, please call UTS at 815-963-2220.

The purpose of the Optimizer is to find the set of input values that would provide the preferred solution to the current model. The user can set preference criteria and limitations to be imposed in the search. The Optimizer can help solve linear, nonlinear, and integer programming problems. It can also be used as an alternative to TK Solver's built-in Iterative Solver for solving systems of equations. In this use the Optimizer offers more control over constraints and bounds on variables.

The Optimizer acts like a shell around a TK Solver model. It calls the model repeatedly, starting with the current values, until the constraints are satisfied or the process fails. It is important to distinguish the Optimizer from TK functions. You cannot include a rule on the Rule Sheet to activate the optimizer. TK does not call the Optimizer. The reverse is actually true: the entire TK model acts like a subroutine called by the Optimizer.

The Optimizer requires that the underlying TK model be solvable without error conditions over the domain of the variables in the optimization. If a TK error is triggered, the Optimizer is halted and the error message is displayed. Additional bounds can be added to prevent such errors. After such an error, the Optimizer must be restarted. It does not continue on from the point of the error.

It is recommended that a model be tested before running the Optimizer on it. Such testing would automatically cause the outputs (or error messages) to appear.

Any TK Solver model can be used with the Optimizer. There is no special programming required. The Optimizer works with the existing variables and lists. This means that all the TK models from the many TK-based applications can be optimized.

Here are the steps required for using the Optimizer:

1. Build a TK model that solves over the required domain.
2. Set up the optimization constraints and bounds using either the Set Up Optimizer Command or the Optimizer Wizard.
3. Press F11 to launch the Optimizer.

Optimize What?

An optimizer is a software tool that help users find the best way to design components or allocate scarce resources. The resources may be raw materials, machine time or people time, money, or anything else in limited supply. The "best" or optimal solution may be maximizing profits, minimizing costs, or achieving the best possible quality. An almost infinite variety of problems can be tackled this way.

The Optimizer significantly broadens the domain of problems you can solve with TK. Students and professionals in all areas of science, engineering, finance, operations research and management will find this to be a valuable, time-saving tool. Here are a few problems TK can now tackle with ease.

Finance and Investment

Working capital management involves allocating cash to different purposes (accounts receivable, inventory, etc.) across multiple time periods, to maximize interest earnings.

Capital budgeting involves allocating funds to projects that initially consume cash but later generate cash, to maximize a firm's return on capital.

Portfolio optimization –creating "efficient portfolios" – involves allocating funds to stocks or bonds to maximize return for a given level of risk, or to minimize risk for a target rate of return.

Manufacturing

Job shop scheduling involves allocating time for work orders on different types of production equipment, to minimize delivery time or maximize equipment utilization.

Blending (of petroleum products, ores, animal feed, etc.) involves allocating and combining raw materials of different types and grades, to meet demand while minimizing costs.

Cutting stock (for lumber, paper, etc.) involves allocating space on large sheets or timbers to be cut into smaller pieces, to meet demand while minimizing waste.

Distribution and Networks

Routing (of goods, natural gas, electricity, digital data, etc.) involves allocating something to different paths through which it can move to various destinations, to minimize costs or maximize throughput.

Loading (of trucks, rail cars, etc.) involves allocating space in vehicles to items of different sizes so as to minimize wasted or unused space.

Scheduling of everything from workers to vehicles and meeting rooms involves allocating capacity to various tasks in order to meet demand while minimizing overall costs.

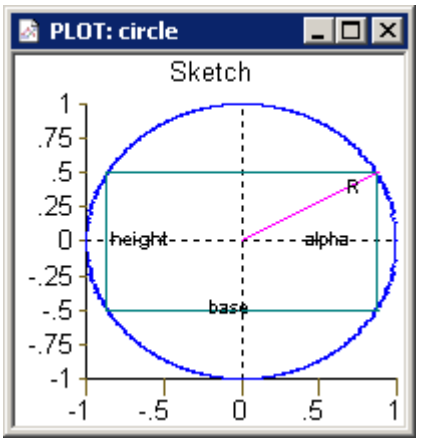
Anything you can think of modeling with TK Solver is now fair game for the Optimizer. Just enter some equations and you're ready to go!

Here are two examples, followed by a summary of the commands and features, and three case studies for additional practice.

1 - Trivial Example

A simple example, with known solution, illustrates the basics of this tool. We will find the rectangular quadrilateral of largest area that can be inscribed in a circle of fixed radius.

We begin by modeling the situation for a sample radius of unit value.



A model of this situation might be the following:

Rules

Status	Rule
Comm	;Simplest.tkw Simplest model for Optimizer demonstration
Comm	;Rectangle of maximum area inscribed in circle of given radius
Satisfi	base = 2 * R * cosd(alpha)
Satisfi	height = 2 * R * sind(alpha)
Satisfi	area = base * height

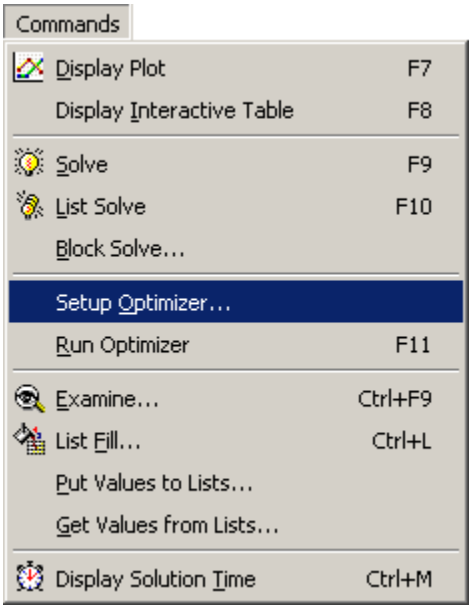
Variables

Status	Input	Name	Output	Unit	Comment
	1	R			
	30	alpha			
		base	1.73205081		
		height	1		
		area	1.73205081		

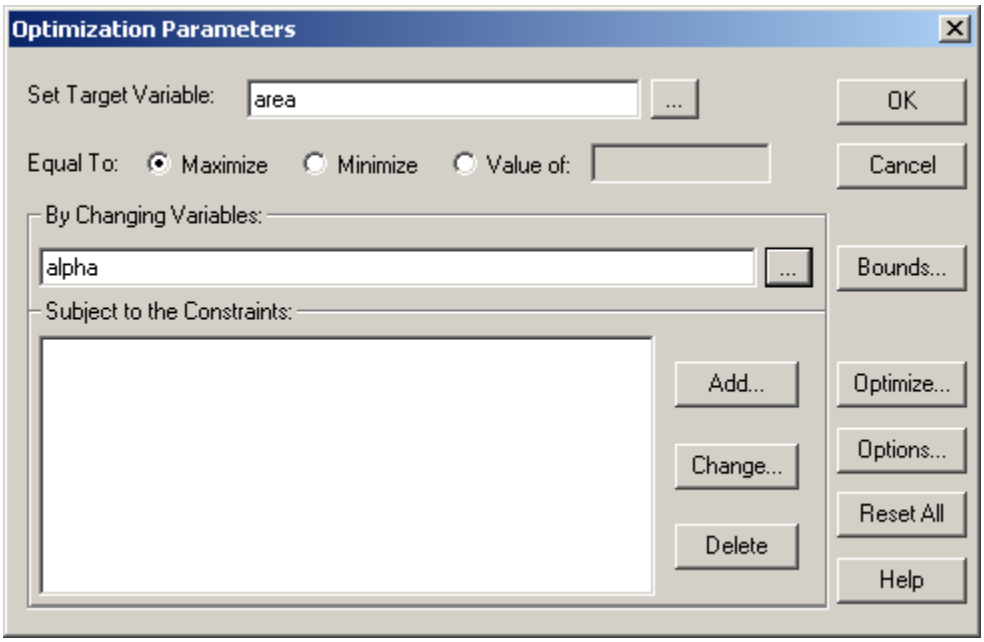
We can see that assigning different input values to the angle `alpha` gives different solutions for the variable `area`. Our goal is to find the value of alpha that results in the maximum value of area.

The Optimizer first searches the Variable Sheet for outputs and provides a list of variables that can be declared as the *target variable*. If the model has not been solved and no outputs are displayed, the Optimizer will not display any target variable options. In this example, the target variable is `area`.

We open the Optimizer. Clicking *Commands*→ *Setup Optimizer*…



The parameters for the optimization can then be set in the panel that opens:



Select the target variable and the variable(s) that the Optimizer will modify (the **decision variables**) by using the selection (…) buttons[1]. We will set the value of "Equal To:" to "Maximize" (the default). No constraints are required for this problem.

Clicking the "Optimize" button causes the model to solve repeatedly under control of the Optimizer, which modifies `alpha` each run until `area` reaches a maximum, 2, at the angle 45 (degrees) as expected.

2 – Typical Example, With Constraints – Using the Optimizer Wizard

The following problem is taken from *Basic Programs for Production and Operation Management* by Pantumsinchai, Hassan, and Gupta (Prentice-Hall, 1983).

Butane, Virgin Naphtha and Catalytic Cracked Gasoline are to be blended into Super and Regular unleaded gasoline. The goal is to blend the components so as to produce the two types of gasoline with a maximum profit. There are six unknowns, corresponding with the daily consumption of components used for each kind of gasoline. This can be represented by two equations, with `TS` and `TR` representing the total units of Super and Regular produced. For example, the variable `x11` represents the units of Butane in Super.

$$TS = x11 + x21 + x31$$
$$TR = x12 + x22 + x32$$

For any blend, the octane rating can be computed for Super and Regular using the following equations, with `OCT1`, `OCT2`, and `OCT3`, the octane rating values of the three components.

$$OCTS = \frac{OCT1 \cdot x11 + OCT2 \cdot x21 + OCT3 \cdot x31}{TS}$$
$$OCTR = \frac{OCT1 \cdot x12 + OCT2 \cdot x22 + OCT3 \cdot x32}{TR}$$

The minimum octane numbers for Super and Regular are 92 and 87 respectively.

Another important criterion of the blends is the Reid Vapor Pressure (RVP). The following equations are used to compute the RVP for Super and Regular, with RVP1, RVP2, and RVP3 the values from each of the components.

$$RVPS = \frac{RVP1 \cdot x11 + RVP2 \cdot x21 + RVP3 \cdot x31}{TS}$$
$$RVPR = \frac{RVP1 \cdot x12 + RVP2 \cdot x22 + RVP3 \cdot x32}{TR}$$

[1] If you do not see the target you had in mind, or if you cannot select the variables you want to adjust, re-check in the Variable Sheet to be certain the target variable has an output value and the variables to be adjusted have input values.

The maximum for both RVPS and RVPR is 8.

There are limitations on the availability of each of the components. The equations for the total of each component used are as follows.

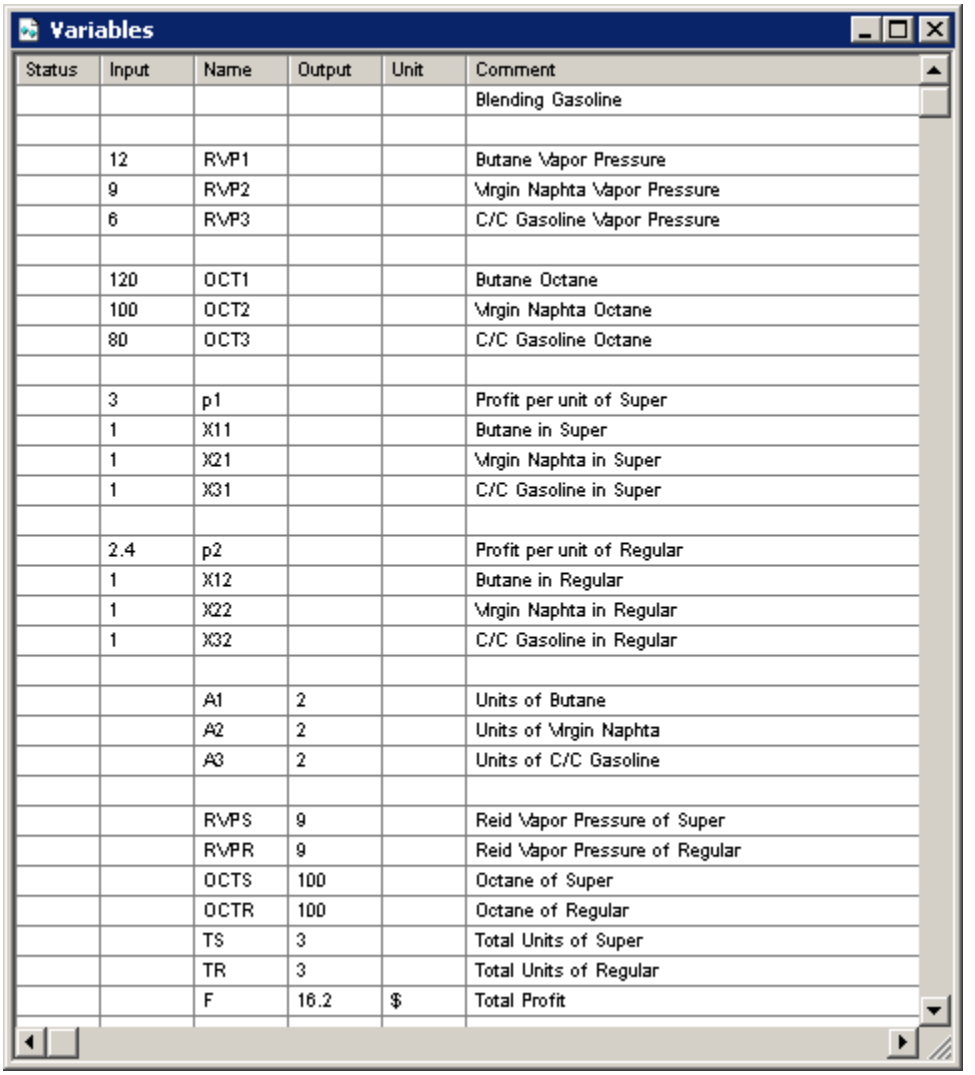
$$A1 = x11 + x12$$
$$A2 = x21 + x22$$
$$A3 = x31 + x32$$

This will allow us to compare A1, A2, and A3 with the available amounts.

The last equation defines the total profit from both blends, where p1 and p2 are the profits per unit of production of Super and Regular, respectively.

$$F = p1 \cdot TS + p2 \cdot TR$$

These equations can be entered into TK Solver to establish the objective function for the optimization. The TK Solver Variable Sheet is shown next.



Variables

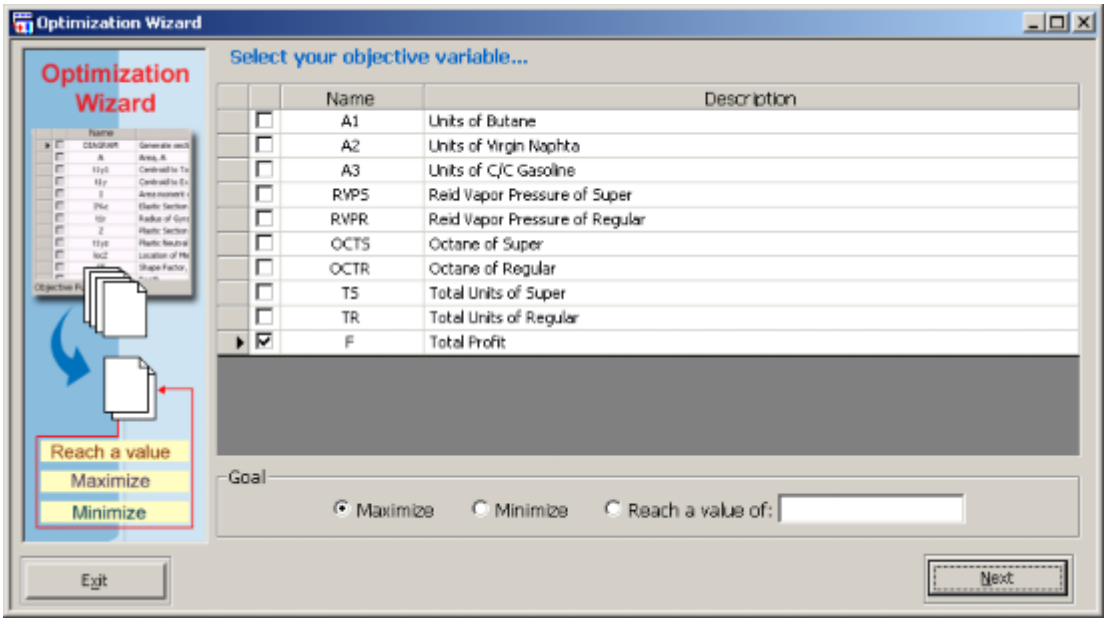
Status	Input	Name	Output	Unit	Comment
					Blending Gasoline
	12	RVP1			Butane Vapor Pressure
	9	RVP2			Virgin Naphta Vapor Pressure
	6	RVP3			C/C Gasoline Vapor Pressure
	120	OCT1			Butane Octane
	100	OCT2			Virgin Naphta Octane
	80	OCT3			C/C Gasoline Octane
	3	p1			Profit per unit of Super
	1	X11			Butane in Super
	1	X21			Virgin Naphta in Super
	1	X31			C/C Gasoline in Super
	2.4	p2			Profit per unit of Regular
	1	X12			Butane in Regular
	1	X22			Virgin Naphta in Regular
	1	X32			C/C Gasoline in Regular
		A1	2		Units of Butane
		A2	2		Units of Virgin Naphta
		A3	2		Units of C/C Gasoline
		RVPS	9		Reid Vapor Pressure of Super
		RVPR	9		Reid Vapor Pressure of Regular
		OCTS	100		Octane of Super
		OCTR	100		Octane of Regular
		TS	3		Total Units of Super
		TR	3		Total Units of Regular
		F	16.2	$	Total Profit

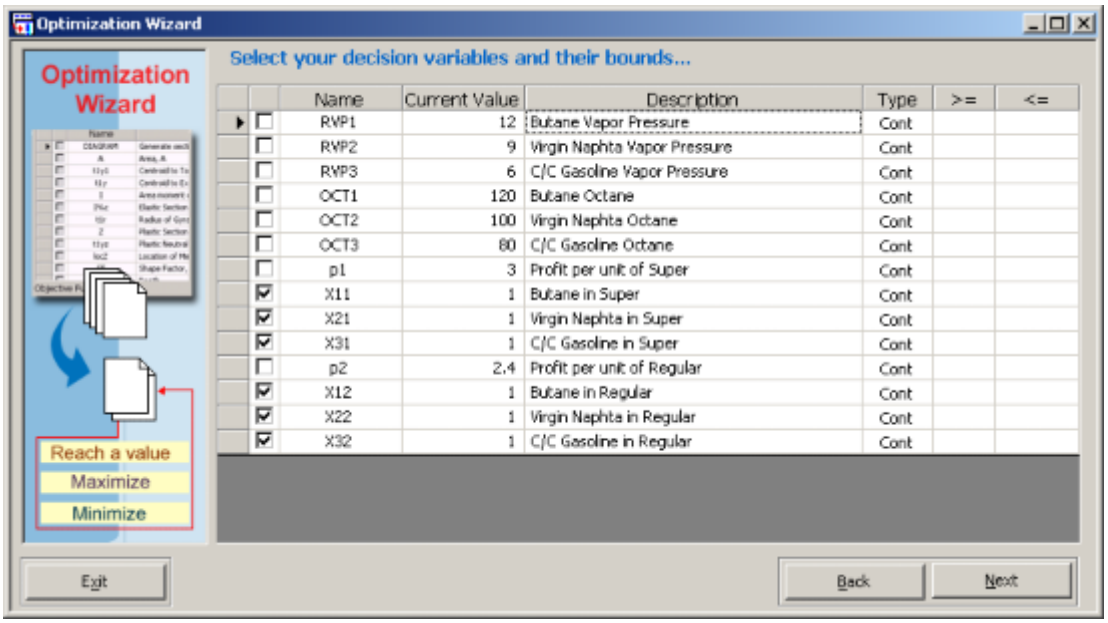
With inputs provided for each component and with values of 1 input for each of the six sample quantities, we see the results at the bottom of the sheet. Our goal is to determine the values of these six quantities that produce a maximum value for `F` and which satisfy the vapor pressure and octane rating requirements for the blends. To accomplish this, we can use the TK Optimizer.

We launch the Optimizer Wizard and define the target variable and constraints.

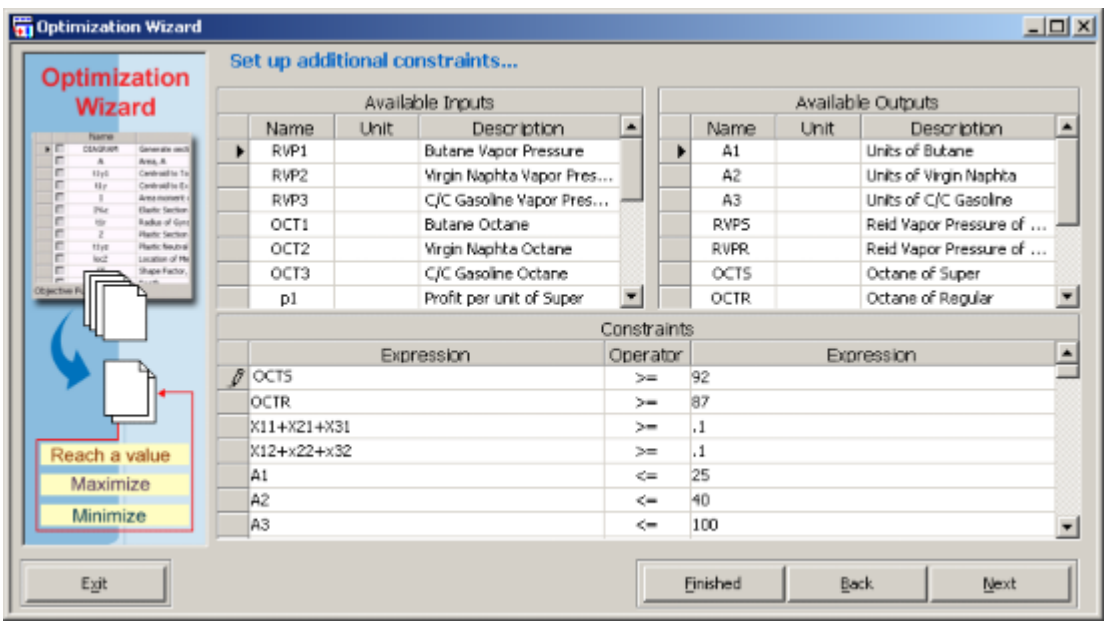
- Total profit, `F`, is selected as the **objective variable** (the same as a target variable).
- Quantities `x11, x21, x31, x12, x22,` and `x32` are identified as decision variables with minimal values set to 0.
- Constraints are placed on the octane rating of Super and Regular, variables `OCTS` and `OCTR`, to be a minimum of 92 and 87, respectively.
- Constraints are placed on the resulting vapor pressures of both the Super and Octane blends. In this case, the maximums are 8 for both.
- Constraints are placed on the available units of each of the three components. 25 units of Butane, 40 units of Virgin Naphtha, and 100 units of C/C Gasoline.
- Constraints are added to indicate that at least some Super and some Regular must be produced. That is, `x11 + x21 + x31 => 0.1` and `x12 + x22 + x32 => 0.1`.

The Optimization Wizard screens are shown on the following pages, along with the optimal solution.





	Name	Current Value	Description	Type	>=	<=
☐	RVP1	12	Butane Vapor Pressure	Cont		
☐	RVP2	9	Virgin Naphta Vapor Pressure	Cont		
☐	RVP3	6	C/C Gasoline Vapor Pressure	Cont		
☐	OCT1	120	Butane Octane	Cont		
☐	OCT2	100	Virgin Naphta Octane	Cont		
☐	OCT3	80	C/C Gasoline Octane	Cont		
☐	p1	3	Profit per unit of Super	Cont		
☑	X11	1	Butane in Super	Cont		
☑	X21	1	Virgin Naphta in Super	Cont		
☑	X31	1	C/C Gasoline in Super	Cont		
☐	p2	2.4	Profit per unit of Regular	Cont		
☑	X12	1	Butane in Regular	Cont		
☑	X22	1	Virgin Naphta in Regular	Cont		
☑	X32	1	C/C Gasoline in Regular	Cont		

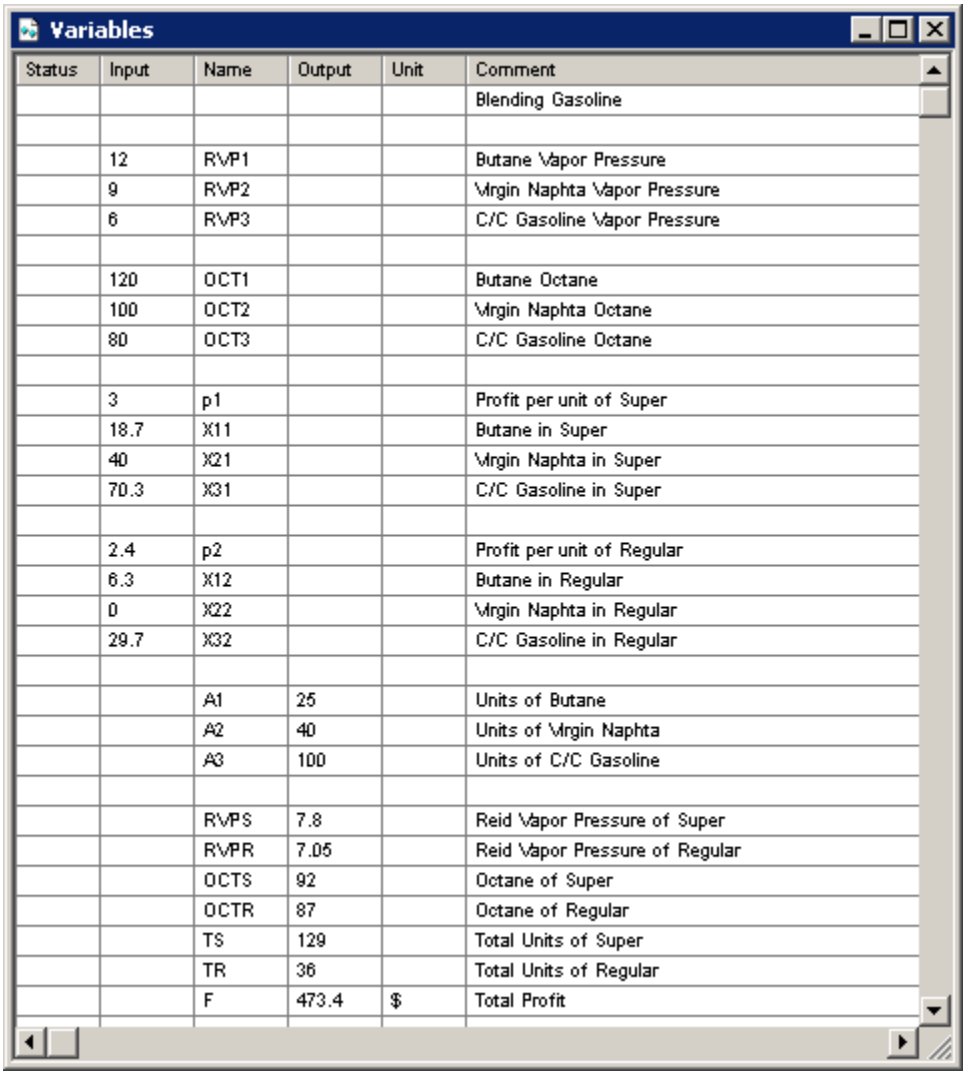


Available Inputs		
Name	Unit	Description
RVP1		Butane Vapor Pressure
RVP2		Virgin Naphta Vapor Pres...
RVP3		C/C Gasoline Vapor Pres...
OCT1		Butane Octane
OCT2		Virgin Naphta Octane
OCT3		C/C Gasoline Octane
p1		Profit per unit of Super

Available Outputs		
Name	Unit	Description
A1		Units of Butane
A2		Units of Virgin Naphta
A3		Units of C/C Gasoline
RVPS		Reid Vapor Pressure of ...
RVPR		Reid Vapor Pressure of ...
OCTS		Octane of Super
OCTR		Octane of Regular

Constraints		
Expression	Operator	Expression
OCTS	>=	92
OCTR	>=	87
X11+X21+X31	>=	.1
X12+x22+x32	>=	.1
A1	<=	25
A2	<=	40
A3	<=	100

Clicking the "Finished" button produces the solution quickly, displayed on the TK Variable Sheet.



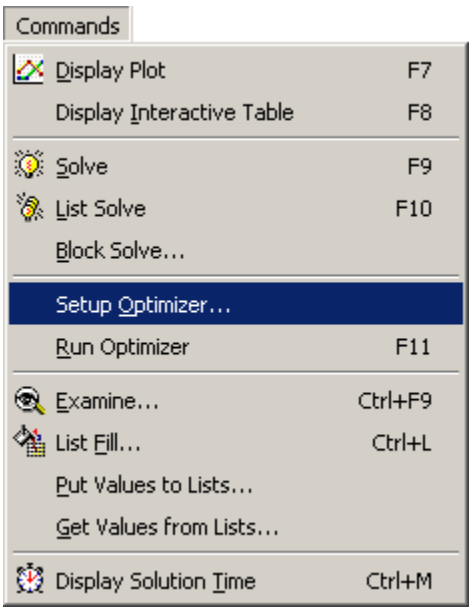
Variables

Status	Input	Name	Output	Unit	Comment
					Blending Gasoline
	12	RVP1			Butane Vapor Pressure
	9	RVP2			Virgin Naphta Vapor Pressure
	6	RVP3			C/C Gasoline Vapor Pressure
	120	OCT1			Butane Octane
	100	OCT2			Virgin Naphta Octane
	80	OCT3			C/C Gasoline Octane
	3	p1			Profit per unit of Super
	18.7	X11			Butane in Super
	40	X21			Virgin Naphta in Super
	70.3	X31			C/C Gasoline in Super
	2.4	p2			Profit per unit of Regular
	6.3	X12			Butane in Regular
	0	X22			Virgin Naphta in Regular
	29.7	X32			C/C Gasoline in Regular
		A1	25		Units of Butane
		A2	40		Units of Virgin Naphta
		A3	100		Units of C/C Gasoline
		RVPS	7.8		Reid Vapor Pressure of Super
		RVPR	7.05		Reid Vapor Pressure of Regular
		OCTS	92		Octane of Super
		OCTR	87		Octane of Regular
		TS	129		Total Units of Super
		TR	36		Total Units of Regular
		F	473.4	$	Total Profit

Setting Up the Optimizer

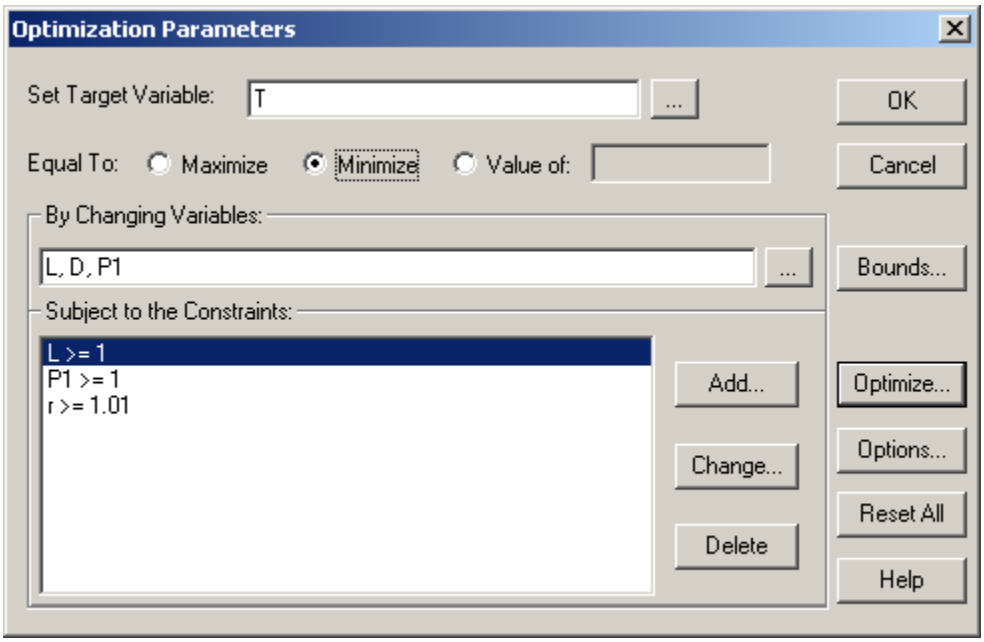
The Optimizer is based on the widely used Solver technology from Frontline Systems. If you are familiar with Microsoft Excel, you might know that it includes a version of Frontline's Solver as an add-in, and you may have become comfortable with that interface to Solver. TK includes two different approaches to setting up the Optimizer. One is very similar to the Excel Solver approach, so Excel users feel comfortable right away. We also include a second approach, via a TK Solver Wizard, which we hope is easier to use for those of you who might be new to the concept.

The TK Solver Commands menu includes two selections related to the Optimizer.



"Setup Optimizer…"—This starts the configuration process, which can culminate with running the Optimizer or saving the settings for later. The interface is similar to that used for configuring the Excel Solver.

"Run Optimizer"—This runs the Optimizer with the values last selected, using current Variable Sheet inputs as initial guesses. It is equivalent to using "Setup Optimizer…" without changing anything. The keystroke shortcut is F11.



The Optimizer works with several different types of variables and it is important to note how it interacts with them.

The **Target Variable** is the TK variable that you want to maximize, minimize, or set equal to a value. Any TK variable that does not have an input value can be set as the target variable. A target variable is sometimes referred to as a **dependent variable** or an **objective variable**.

The target can be any valid TK expression that produces a numeric value. If the target value in a TK model is actually the element of a list, you can create a rule equating the list element with a variable before using the Optimizer. For example, if you want to use the 100th element of the list named `temperature` as your target value, you could create the following rule on the Rule Sheet and use variable `tn` as your target variable.

```
tn = 'temperature[100]
```

Change Variables are those variables that you want the Optimizer to try changing in order to achieve your goal for the target variable. A change variable must be an input variable in the TK model. It cannot have a guess status. Change variables are sometimes called **decision variables** or **independent variables**.

If a change value of interest in a TK model is actually a list element, you must create a rule placing the value of a variable from the Variable Sheet into the list element before using the Optimizer. For example, if the 4th element of the list named depth should be changed in the optimization process, the following rule assigns the value of the variable named D to the list element and the Optimizer uses D as a change variable.

```
Place('depth,4) = D
```

It is very important that you **not use a TK variable with a guess status for a target value** when you launch the Optimizer. A guess value will only be used by the Optimizer on its first pass through the TK model and will not be available on subsequent passes. If a guess value is required in the model, use a first guess on the variable subsheet; that guess will be triggered automatically on each pass. Such a variable could be used as the target variable by the Optimizer.

Available Variables

Whenever you see the ellipsis button you may click on it for a list of variables to add to the current field. The variables listed will meet the required criteria. For example, if you click on the ellipsis button for the Target Variable, you will be shown a list of variables with output status on the TK Variable Sheet. Likewise, the ellipsis button for the change variables produces a list of variables that have inputs.

Bounds and Constraints

Bounds are limits beyond which the Optimizer should not search. **Constraints** are limitations set on the end result, but the Optimizer may search beyond them in finding the optimal solution. In other words, it may be necessary for the Optimizer to get a bigger picture of the problem to better solve it. Because of this important difference, bounds and constraints are handled separately in the interface.

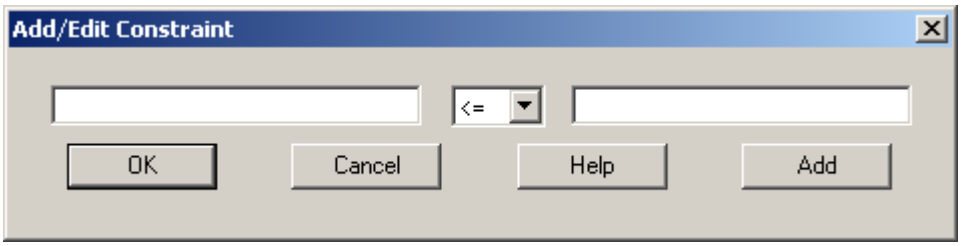
It is possible to set bounds for variables that may cause an error condition. For example, if you know that the model fails if the value of `T` exceeds 100, and `T` is one of your change variables, you can set an upper bound on `T`. You can also set bounds that are functions of existing variables and expressions, but it is very important to note that these bounds are set based on the initial conditions of the model and do not dynamically update as the model is repeatedly solved. Bounds should be numeric values or expressions involving only input variables or list elements that will not change in the solution process. For example, you might set a bound that `Di` must be less than `Do`, restricting an inner diameter to be less than an outer diameter. This allows you to change the input for `Do` and rerun the optimization without having to use the commands menu to change the bounds.

Optimizer can violate the bounds but will only do so as a last resort if it is unable to find its way to a solution within the bounds.

Constraints can look exactly the same as bounds, but constraints are repeatedly updated and checked, and they can be temporarily violated during the course of the optimization process. For example, if you have a variable `p` whose result should not be more than 80% of `pcrit`, you can set a constraint, `p <= 0.8*pcrit`, and the Optimizer will use that as a guide. At times during the process, `p` might exceed the limit, but the Optimizer will not finish until the constraint is satisfied. When all the constraints are true and the Optimizer fails to improve on the target condition, the Optimizer is finished and the results are posted.

Adding Constraints

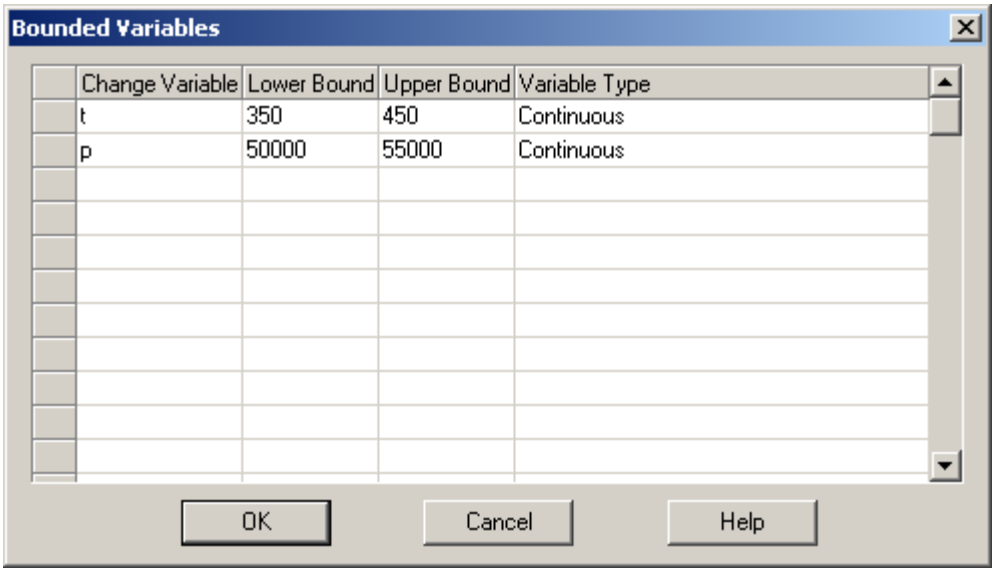
Clicking the Add button opens the Add/Edit Constraint dialog for entry of constraints. Any valid TK expressions are allowed. This dialog will also appear if you double-click on any existing constraints in the Optimization Parameters window. In that case, the constraint will be displayed and the contents can be edited. An alternative to double-clicking on a constraint is to highlight the constraint and click the Change button. To remove a constraint, highlight it and click the Delete button.



After the expressions have been entered in the Add/Edit Constraint dialog, click the Add button to add another constraint or the OK button to close the dialog and return to the Optimization Parameters window. The Cancel button cancels any edits and returns to the main window.

Adding Bounds

To add bounds on any change variables, click the Bounds button. This opens the Bounded Variables dialog. By default, all the change variables are listed without bounds and declared as continuous type variables. Other Type options are Integer and Binary; you can change the type to either of those by entering an "i" or a "b" in that column. Integer variables can only take on integer values. Binary variables can only be 0 or 1. Bounds can be expressions based on other inputs in the model. If `t` must be greater than `Tcrit`, then `Tcrit` can be assigned as the lower bound. You should not use change variables as bounds for other change variables. The bound values are only evaluated at the beginning of the optimization process.

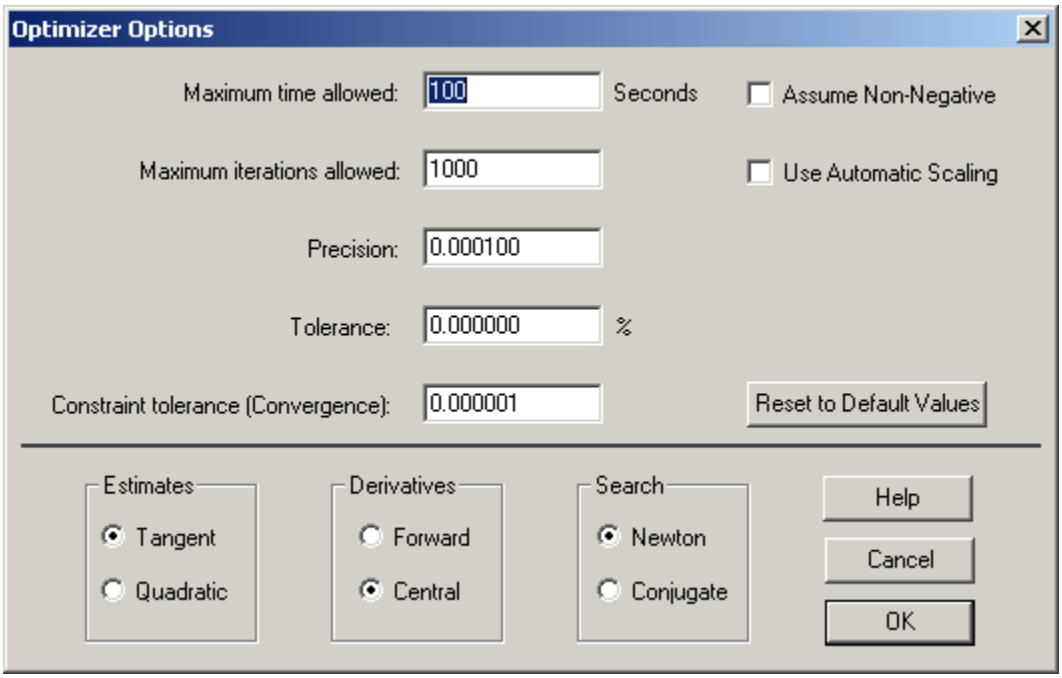


Optimize

The Optimize… button launches the Optimizer. This can also be done using function key F11. It may be useful to return to the Variable Sheet before launching the Optimizer so you can watch the action.

Options

Clicking the Options button opens the Options screen.



Maximum time allowed defaults to 100 seconds. You may want to extend this for problems that you know will take a long time to solve. The limit is 32000.

Maximum iterations allowed defaults to 1000. You may want to extend this for problems that you know require many iterations. The limit is 32000.

Precision controls the precision of solutions by using the number you enter to determine whether the value of a constraint cell meets a target or satisfies a lower or upper bound. Precision must be indicated with a fractional number between 0 and 1. Higher precision is indicated when the number you enter has more decimal places; for example, 0.0001 is higher precision than 0.01. The default is 0.0001.

Tolerance is used to compare values with the closest integer when integer variable bounds are used.

Constraint Tolerance (Convergence)—When the relative change in the target cell value is less than the number in the Convergence box for the last five iterations, TK stops. Convergence applies only to nonlinear problems and must be indicated by a fractional number between 0 (zero) and 1. A smaller convergence is indicated when the number you enter has more decimal places; for example, 0.0001 is less relative change than 0.01. The smaller the convergence value, the more time TK takes to reach a solution.

Assume Non-Negative is an option you may want to use frequently. It saves you the trouble of setting bounds and/or constraints on all variables that you want to assume are greater than or equal to zero.

Use Automatic Scaling —Select this to use automatic scaling when inputs and outputs have large differences in magnitude.

Estimates—This specifies the approach used to obtain initial estimates of the basic variables in each one-dimensional search. The *Tangent* option uses linear extrapolation from a tangent vector. *Quadratic* uses quadratic extrapolation, which can improve the results for highly nonlinear problems.

Derivatives —This specifies the differencing used to estimate partial derivatives of the objective and constraint functions. Use the *Forward* option for most problems, in which the constraint values change relatively slowly. *Central* should be used for problems in which the constraints change rapidly, especially near the limits. Although this option requires more calculations, it might help when TK returns a message that it could not improve the solution.

Search —This specifies the algorithm that is used at each iteration to determine the direction to search. *Newton* uses a quasi-Newton method that typically requires more memory but fewer iterations than the Conjugate gradient method. *Conjugate* requires less memory than the Newton method but typically needs more iterations to reach a particular level of accuracy. Use this option when you have a large problem and memory usage is a concern, or when stepping through iterations reveals slow progress.

Units and the Optimizer

The Optimizer works with variables in their calculation units. This is done to be consistent with the rules on the rule sheet and makes TK-based reports easier to understand. It also means that users can subsequently change the display units for any variables without worrying about affecting any Optimizer settings.

You may want to set all the variables directly involved in the optimization to their calculation units before initially setting the optimization constraints and bounds. This will make it easier to make your optimization entries and to compare values with the bounds and constraints. Use the Properties dialog to quickly view the calculation units for each variable of interest.

Users of existing TK models should be careful in preparing new Optimization settings, just as they must be careful when adding new rules to a model. In both situations, it is vital that the calculations are done in consistent units.

Units and Improved Performance and Accuracy

You can use units and conversions to improve the performance of the Optimizer. Try to use display units that cause values of variables to be relatively close to 1. Since the default precision and comparison tolerance values are around 1E-6, it can be helpful to try to keep absolute values between 1E-6 and 1E6 but closer to 1 is always better. If the target variables, the change variables or any of the variables involved in the constraints are relatively small or relatively large, it might be causing the Optimizer to work harder to reach a solution. On rare occasions, the Optimizer might even "overlook" the solution altogether.

Using the pressure units example above, it's possible that your equations are given in units of Pa. The resulting `pcrit` value would be 60000000 Pa. By converting to MPa, the `pcrit` value becomes 60 and the Optimizer can work with it more efficiently.

For more on computational accuracy, see the Technical Time Out starting on page 5-13.

Optimization with Lists and Tables

TK Solver includes a flexible programming language for processing lists and matrices. Values in lists can be used as inputs or outputs in optimization problems by mapping them with variables on the Variable Sheet.

Change Variables

A change variable used in an optimization process can be placed into a list using the PLACE function. For example, the following rule can be used to place the value of the variable named `throttle` into the 10th element of the list named `xyz`.

```
PLACE('xyz,10) = throttle
```

Target Variables

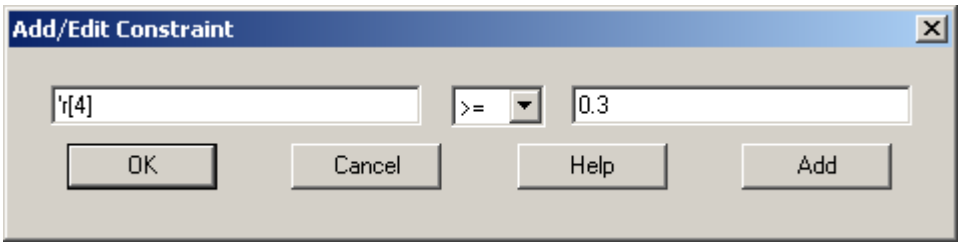
A target variable can receive a value from a list directly on the Rule Sheet. The rule below sets the value of the variable `T` equal to the value of the nth element of the list `abc`.

```
T = 'abc[n]
```

It is important to note the difference between referencing list elements in this way and using the PLACE function. Both are one-way processes, but in opposite directions.

Constraints

Constraints can reference list elements directly. For example, you can add a constraint that specifies that the 4th element of the list r must be greater than or equal to 0.3.



Troubleshooting

The Optimizer might stop before reaching a solution for any of the following reasons:

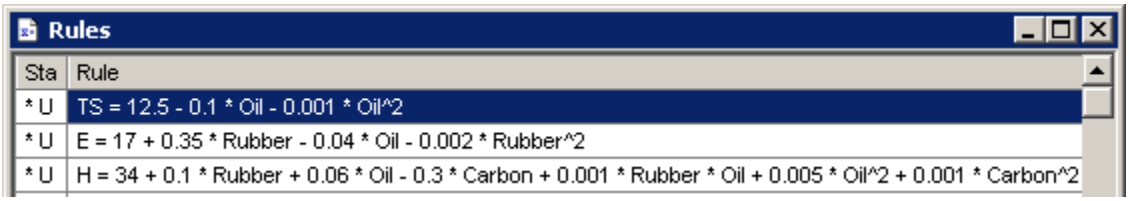
1. You interrupted the solution process by pressing Ctrl-Break.
2. The maximum time or number of iterations was reached.
3. The Target Variable or Objective Function you specified is increasing or decreasing without limit.
4. For problems with integer constraints, you need to decrease the Tolerance setting in the Options dialog box so that Optimizer can find a better integer solution.
5. For nonlinear problems, you need to decrease the Convergence setting in the Options dialog box so that Optimizer can keep searching for a solution when the target variable value is changing slowly.
6. You need to select the Use Automatic Scaling check box in the Options dialog box because some input values are several orders of magnitude apart, or input and output values differ by several orders of magnitude.
7. Errors occurred in the TK model. You may need to add bounds to avoid such error conditions.
8. You used "guess status" variables in the TK model. Use default first guesses on the variable subsheet.
9. Optimizer reports that it is converging too slowly. Try reducing the required precision settings.
10. One or more expressions result in 0/0 divisions, which TK interprets as undefined. When this happens, TK skips the rule. This can cause the optimization process to lose continuity and to behave erratically. Add constraints to the model to force the Optimizer away from such conditions.

More Examples

We conclude this section on the Optimizer with four case studies. The first example demonstrates how to minimize cost. The second solves a portfolio optimization problem involving the use of data tables. The third solves a relatively difficult problem involving the solution of two simultaneous differential equations. The fourth is a simpler problem showing how to store solutions in tables. The Optimization folder in the Mathematics section of the TK Library includes models associated with these examples and more.

Minimize Cost Example

A company produces rubber used for tires by combining three ingredients: rubber, oil, and carbon black. The cost of rubber is 0.04 per pound. The cost of oil is 0.01 per pound. The cost of carbon black is 0.07 per pound. Here are the equations relating the composition with its characteristics.


Rules
Sta
Rule
* U
TS = 12.5 - 0.1 * Oil - 0.001 * Oil^2
* U
E = 17 + 0.35 * Rubber - 0.04 * Oil - 0.002 * Rubber^2
* U
H = 34 + 0.1 * Rubber + 0.06 * Oil - 0.3 * Carbon + 0.001 * Rubber * Oil + 0.005 * Oil^2 + 0.001 * Carbon^2

The hardness must be between 25 and 35, the elasticity must be at least 16. The tensile strength must be at least 12.

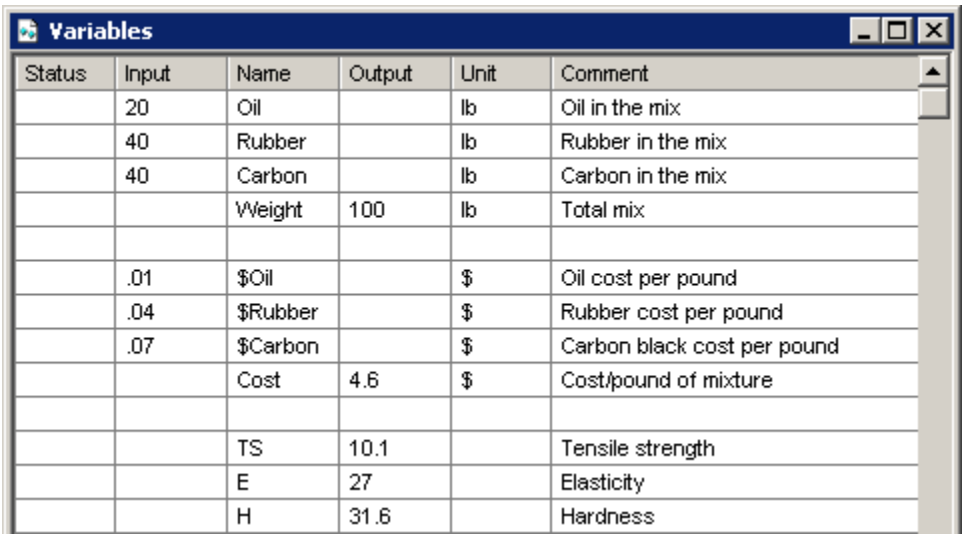
Here is the equation for the cost of the mixture.

```
$Oil*Oil + $Rubber*Rubber + $Carbon*Carbon = Cost
```

Another equation is added to compute the total pounds in the mixture.

```
Weight = Oil + Rubber + Carbon
```

Here is the Variable Sheet with some sample inputs and outputs.



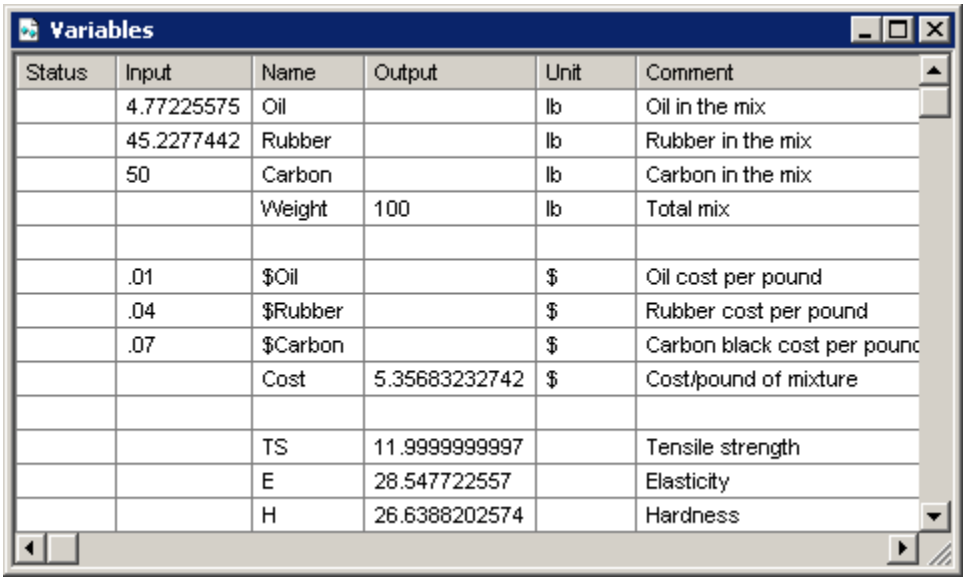
Status	Input	Name	Output	Unit	Comment
	20	Oil		lb	Oil in the mix
	40	Rubber		lb	Rubber in the mix
	40	Carbon		lb	Carbon in the mix
		Weight	100	lb	Total mix
	.01	$Oil		$	Oil cost per pound
	.04	$Rubber		$	Rubber cost per pound
	.07	$Carbon		$	Carbon black cost per pound
		Cost	4.6	$	Cost/pound of mixture
		TS	10.1		Tensile strength
		E	27		Elasticity
		H	31.6		Hardness

Using the equations and variables above, the model will now solve for the cost of the material for a given mixture. The Optimizer can be set up to solve for the mixture producing the required qualities at the lowest cost.

The target variable is `Cost`, and it should be minimized. The change variables are `Oil`, `Rubber`, and `Carbon`, with lower bounds set as 0, 25, and 50. Upper bounds are set as 100, 60, and 100. These bounds are based on the assumption that `Weight` is 100. Here are the constraints.

```
Weight = 100
TS => 12
E => 16
H => 25
H <= 35
```

The Optimizer (F11) returns the following solution.



Variables

Status	Input	Name	Output	Unit	Comment
	4.77225575	Oil		lb	Oil in the mix
	45.2277442	Rubber		lb	Rubber in the mix
	50	Carbon		lb	Carbon in the mix
		Weight	100	lb	Total mix
	.01	$Oil		$	Oil cost per pound
	.04	$Rubber		$	Rubber cost per pound
	.07	$Carbon		$	Carbon black cost per pound
		Cost	5.35683232742	$	Cost/pound of mixture
		TS	11.9999999997		Tensile strength
		E	28.547722557		Elasticity
		H	26.6388202574		Hardness

This example model is located in the Optimization section of the TK Library, in the Optimizer Examples group.

Portfolio Optimization: The Scenario Approach

Given a collection of scenarios and the probabilities associated with them, we can determine the configuration resulting in the optimal outcome. This approach can be used in a variety of situations, including investment portfolio management.

This TK table summarizes seven scenarios for a portfolio of three investments. The values are the expected decimal fraction returns over the next 12 months. For example, scenario 3 predicts that the second investment will return 32.1% over the next year.

INT TABLE: Investments

List	1	2	3
s1	-.071	.144	.169
s2	.056	.107	-.035
s3	.038	.321	.133
s4	.089	.305	.732
s5	.09	.195	.021
s6	.083	.39	.131
s7	.035	-.072	.006

LIST: P

Element	Value
1	.05
2	.1
3	.2
4	.3
5	.2
6	.1
7	.05

The probabilities associated with each of these scenarios are contained in the following list.

The expected return from each of these scenarios can be computed if we input the fraction we invest in each of the three equities. We can expand our input table to include these fractions.

INT TABLE: Investments

List	1	2	3
s1	-.071	.144	.169
s2	.056	.107	-.035
s3	.038	.321	.133
s4	.089	.305	.732
s5	.09	.195	.021
s6	.083	.39	.131
s7	.035	-.072	.006
F	.4	.4	.2

For each of the scenarios, the expected return is computed as the "dot product" of the returns list and the list of the fraction invested. In TK terms, this is represented as `r = DOT('s1,'F)`. Since we need to do this for each of the seven scenarios, we can use a procedure function. We create the function and call it port. Here are the required statements.

PROCEDURE: port

Comment: Portfolio evaluation

St	Statement
C	; Expected returns for each scenario
	m = length('s)
	for i = 1 to m
	'r[i] = dot('F,'s[i])
	next i

These statements assume that a TK matrix, `s`, has been created. The list `s` is simply a list of list names.

LIST: s

Element	Value
1	's1
2	's2
3	's3
4	's4
5	's5
6	's6
7	's7

Now we can call the port function from the Rule Sheet.

Rules

Status	Rule
* Unsati	call port()

The resulting list of `r` values can be included in a table with the corresponding probabilities. These are the probabilities and returns associated with each of the seven scenarios. Note that the sum of the probabilities is 1.

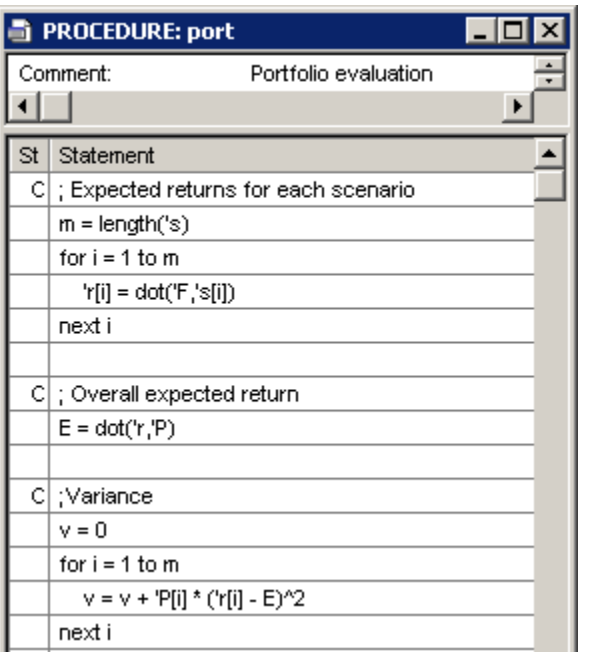
INT TABLE: Probabilities

Element	P	r
1	.05	.063
2	.1	.0582
3	.2	.1702
4	.3	.304
5	.2	.1182
6	.1	.2154
7	.05	-.0136

It is now possible to compute the overall expected return, multiplying the expected returns by the associated probabilities, and summing the results. This is another job for the DOT function. The required statement is added to the `port` procedure function.

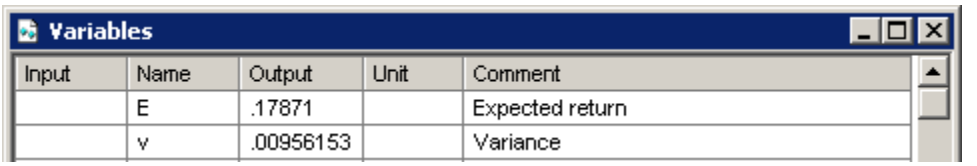
```
E = dot('r,'P)
```

The variance of the returns is then computed using another loop within the procedure. For each scenario, the overall expected return is subtracted from each of the expected scenario returns. That difference is squared and the result multiplied by the associated probability. The results are then summed. The required statements are added to the `port` procedure function. Here is the complete function.



The local variables `E` and `v` are declared as Output Variables in the function and the rule is modified to accept the change.

Solving (F9), the Variable Sheet displays the results for the collection of inputs in the tables.



A common goal in investing is to obtain a certain expected level of return with a minimum variance. In other words, there may be many ways of obtaining a particular level of return but those that are more likely to result in greater fluctuations from the expectation are sometimes less desirable. In this case, we are interested in determining the fraction to invest in each of the equities, such that the return is at least 15%, with minimum variance.

The target value for this problem is the variance, v. The change values are the three fractions in the list F. The Optimizer requires that the change values be declared as variables so additional rules are required. The PLACE function is used to place values from the Variable Sheet into list elements.

Rules

Status	Rule
* Unsati	place('F,1) = F1
* Unsati	place('F,2) = F2
* Unsati	place('F,3) = F3
* Unsati	call port(;E,v)

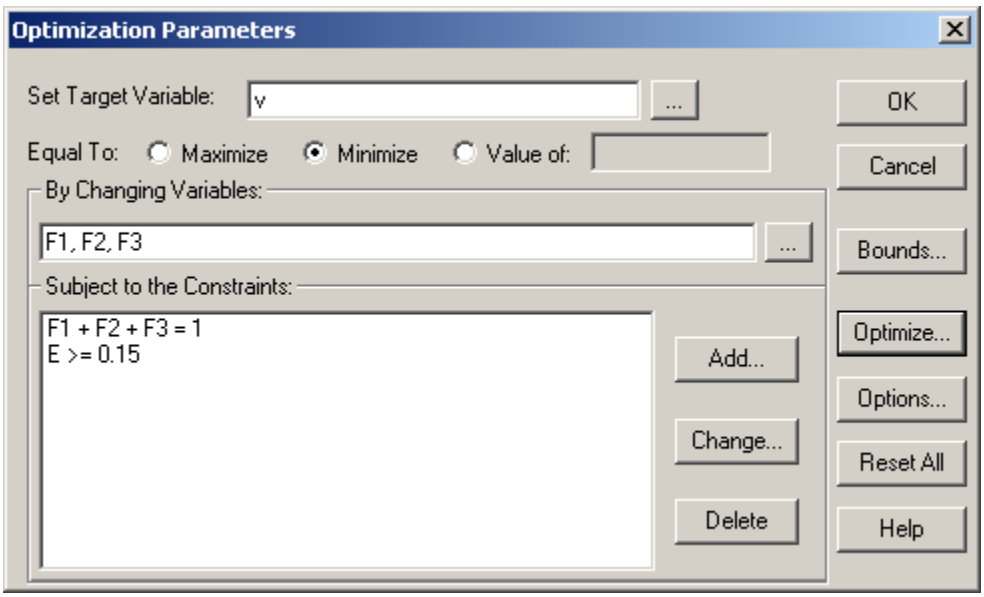
It is important to note the sequence of the rules. The port function assumes that the values have already been placed into the F list, so the function call should follow the rules that perform that task.

The Variable Sheet is updated as shown below.

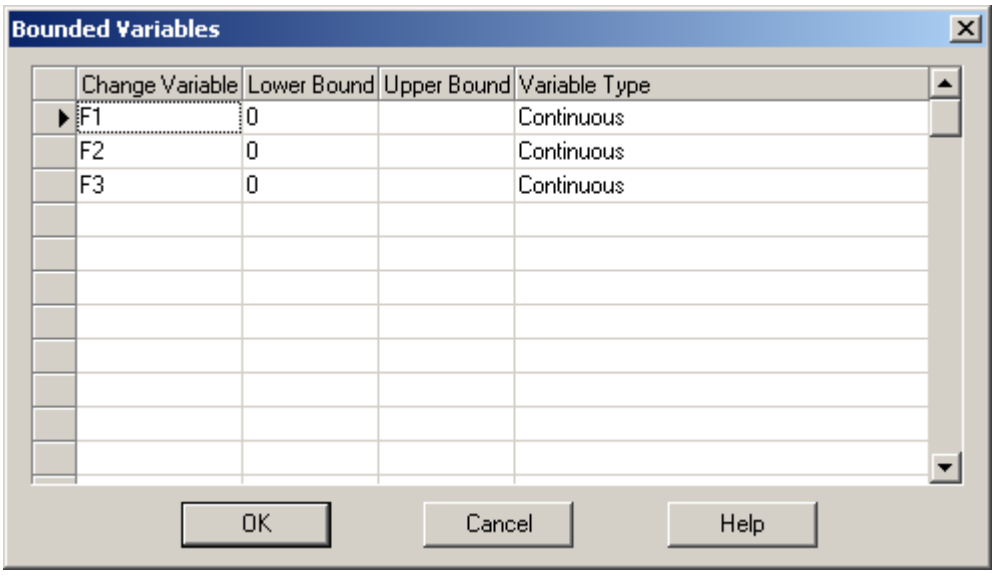
Variables

Status	Input	Name	Output	Unit	Comment
	.4	F1			Fraction of portfolio in equality 1
	.4	F2			Fraction of portfolio in equality 2
	.2	F3			Fraction of portfolio in equality 3
		E	.17871		Expected return
		v	.00956153		Variance

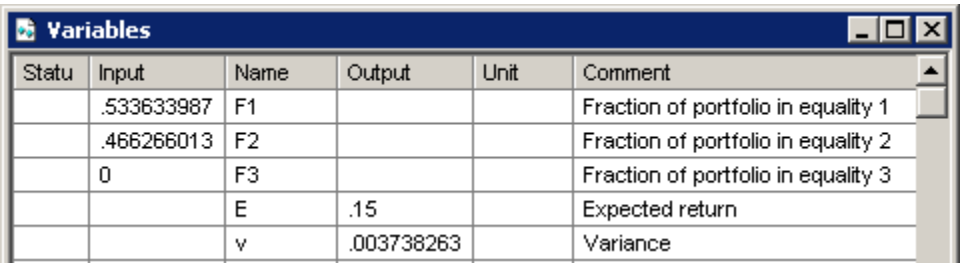
The Optimizer can now be set up to solve the problem.



Constraints are added. We assume that the fractions must total 1 and that the minimum return is 15%. Bounds are also placed on the three change variables, forcing them to be equal to or greater than zero. That is, short selling is not considered.



After a few iterations, the solution is displayed on the Variable Sheet. You might investigate what happens as the minimum expected return is changed, as well as what happens if the bounds are removed.



Differential Equations and the Optimizer

This is a very rigorous exercising of many different TK Solver features, starting with the solution of differential equations and concluding with use of the Optimizer to backsolve the problem.

Suppose one needs to solve the following differential equations for `x = 0.1, 0.2, … 1.6`, with initial conditions `y(0.1) = 0.2 and z(0.1) = 1.0`:

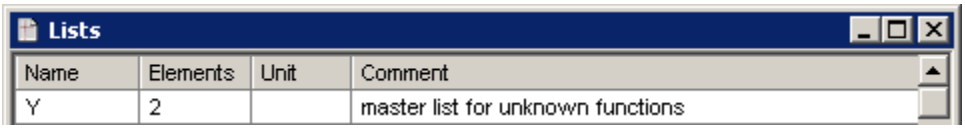
$$y` = 10 \cdot e^{(-2.5 \cdot x)} - 2 \cdot y$$

$$z` = x - \frac{1}{y}$$

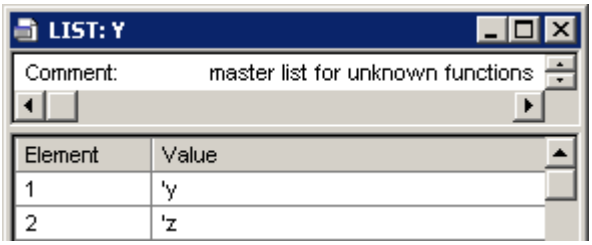
TK Solver includes several built-in functions for solving differential equations numerically. They are all used similarly. For this example, the ODE_RK4 function is used.

The ODE_RK4 function requires three inputs. The first represents the differential equations. If there are simultaneous differential equations, a TK procedure function must be used to define the set. For this example, a function called `test2` is used to hold the equations. The second input to ODE_RK4 is the reference to the list containing the values of the independent variable. These are the values at which the differential equations are to be integrated. The list `x` is used here. The last input to ODE_RK4 is the name of the list containing the solution list names. In this case, we will have two solution lists because we have two differential equations. The list `Y` will be used to store the two solution list names.

The next step is to create the list `Y` on the List Sheet and fill it with the solution list names. The names y and z are used for the two solution lists.



Here are the contents of the list `Y`.



A table can be used to store the independent variable and solution values.

Here are the definitions of the content lists for the table `Solution`.

TABLE: Solution

List	Format	Width	Heading
x		10	
y		10	
z		10	

The table is used to supply the values of the independent variable and the initial conditions for `y` and `z`.

INT TABLE: Solution

Element	x	y	z
1	.1	.2	.1
2	.11		
3	.12		
4	.13		
5	.14		
6	.15		

The List Fill Command is used to fill the `x` list with values from 0.1 to 1.6 in steps of 0.1.

The next step is to create the TK procedure function `test2`, which defines the set of simultaneous differential equations. Here are the statements required.

PROCEDURE: test2

St	Statement
	y`[1] = 10 * exp(-2.5*x) - 2*y[1]
	y`[2] = x - 1/y[1]

The variable `x` is the independent variable. The variable `y`` represents the derivative with respect to the independent variable `x`. The variable `y` represents the function with respect to `x`. The expression `y`[i]` represents the derivative of the `ith` function with respect to `x`. Likewise, the expression `y[i]` represents the value of the `ith` function at the value of `x`.

The variables are mapped to the calling ODE_RK4 function as follows.

PROCEDURE: test2

Comment:	Simultaneous Differential Equations
Parameter Variables:	
Input Variables:	y`,y,x
Output Variables:	

The input variables must be defined in the proper sequence in order for the built-in ODE_RK4 function to work properly. The sequence must represent the derivative, function, and independent variable values respectively. Because these are passed into the function they are declared as Input Variables.

The program is now ready to solve the problem. Click the solve icon and the solution values are displayed in the table. Here are the first 6 elements of the table.

INT TABLE: Solution

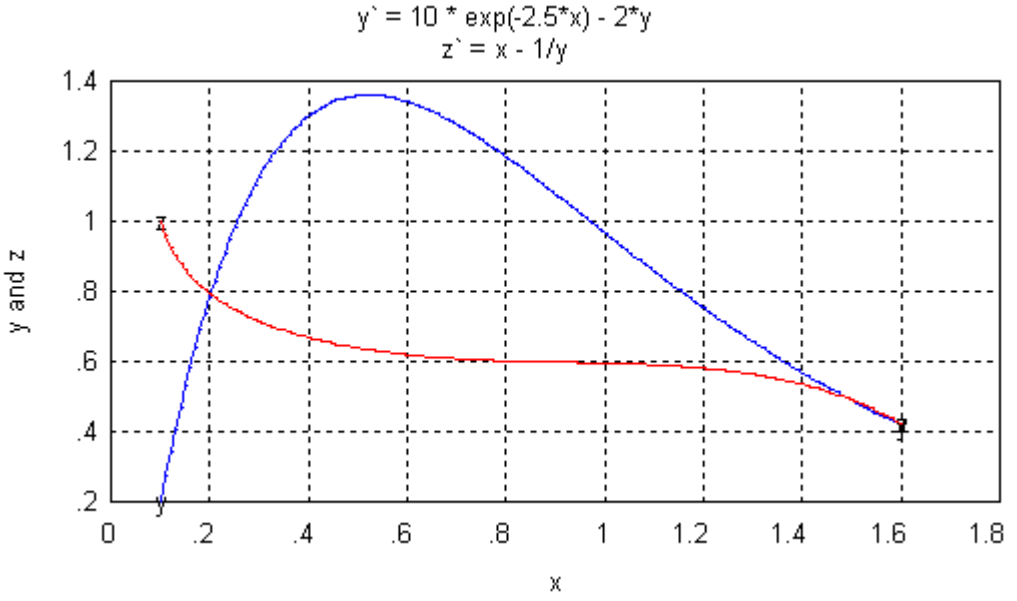
Element	x	y	z
1	.1	.2	1
2	.11	.272187157	.958406162
3	.12	.341064825	.926832997
4	.13	.406744958	.90128761
5	.14	.469336143	.87978279
6	.15	.528943701	.861184223

Here are the last few elements.

INT TABLE: Solution

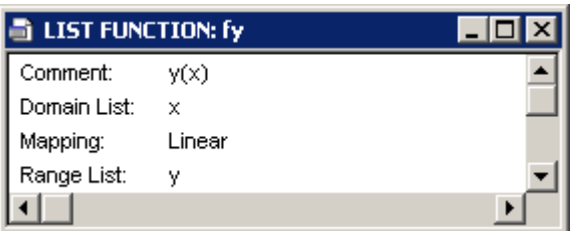
Element	x	y	z
146	1.55	.452960424	.459690414
147	1.56	.446020466	.452992197
148	1.57	.439167827	.446047312
149	1.58	.432402013	.438849371
150	1.59	.425722512	.43139187
151	1.6	.419128792	.423668185

We can easily create a TK Line Chart to plot y and z vs. x.



If these solutions are to be used in other parts of the model, they can be applied in TK List Functions.

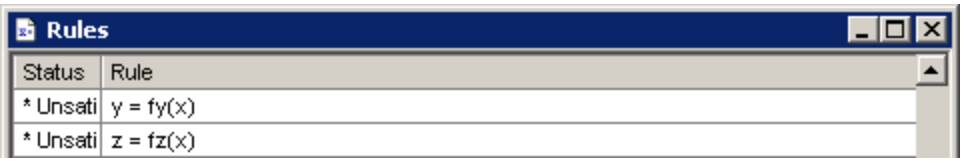
Here is the definition of the y function. The x list is used as the domain of the function. The y list is used as the range list. Using linear interpolation, the function will be continuous over the entire domain of x. The function will be undefined for values of x outside the domain.



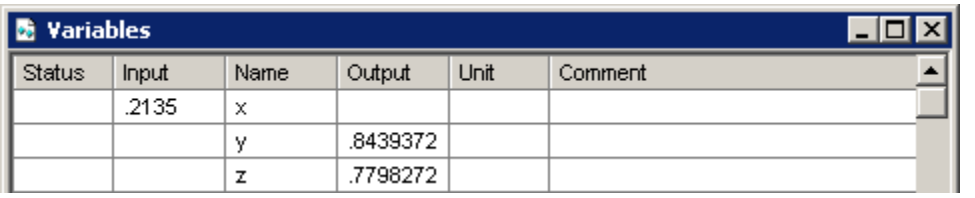
Here are a few elements of the values used in the function.

Element	Domain	Range
1	.1	.2
2	.11	.272187156689373
3	.12	.341064825359446
4	.13	.406744957765867
5	.14	.469336142769403
6	.15	.528943701223073

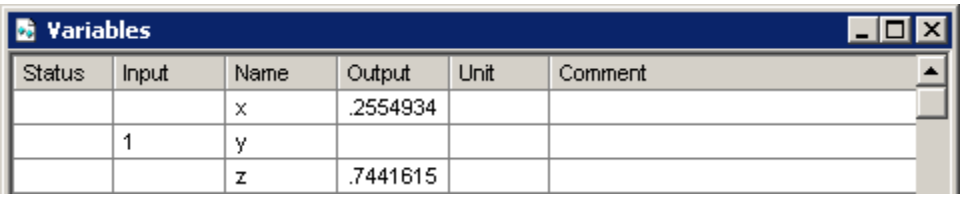
The fz function is set up in the same way. We can now reference these functions anywhere else in the model. Here are two rules on the rule sheet.



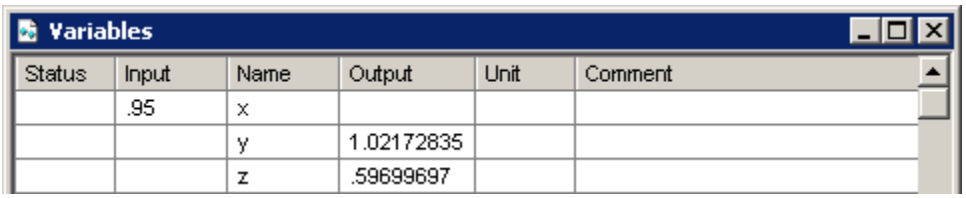
If we enter a value of 0.2135 for x on the Variable Sheet, we can solve for y and z.



We can also input a value for y and solve for x.



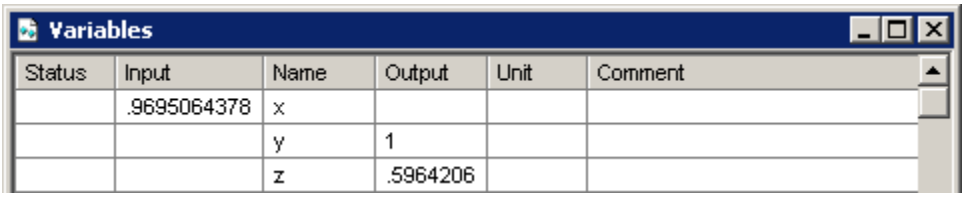
It is important to note that TK List Functions backsolve directly and always return the first solution they find. In this case, there are two solutions as can be seen from the solution plot. It looks like there should be another solution somewhere near `x = 0.95`.



Variables

Status	Input	Name	Output	Unit	Comment
	.95	x			
		y	1.02172835		
		z	.59699697		

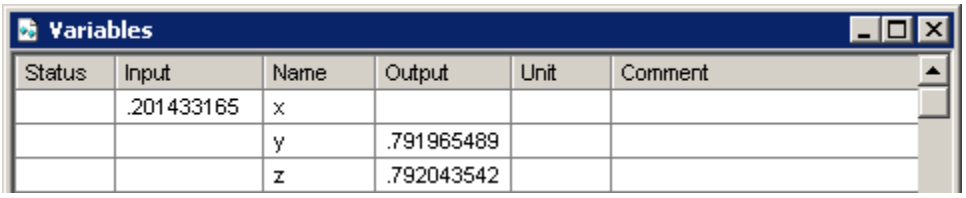
TK Optimizer can be used to find this solution. The Optimizer is configured with the Target Variable defined as `y`, set equal to 1, and with the Change Variable set as `x`. Using bounds of 0.8 and 1 for `x`, the Optimizer displays the following solution.



Variables

Status	Input	Name	Output	Unit	Comment
	.9695064378	x			
		y	1		
		z	.5964206		

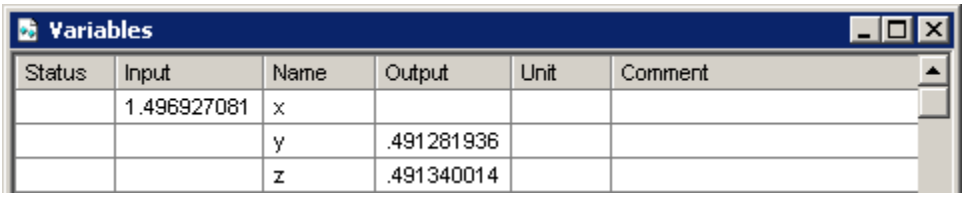
The Optimizer can also be used to determine an intersection point for the `y` and `z` curves. We enter a rule, `d=y-z`, to define the difference between the two functions. We set the objective function as `d` with a target value of 0. Bounds are set on `x` of 0.1 and 0.3 and the Optimizer returns the following solution.



Variables

Status	Input	Name	Output	Unit	Comment
	.201433165	x			
		y	.791965489		
		z	.792043542		

The second intersection point can be found by setting the bounds on `x` of 1.4 to 1.6.



Variables

Status	Input	Name	Output	Unit	Comment
	1.496927081	x			
		y	.491281936		
		z	.491340014		

The Optimizer can also be used to backsolve the differential equations for an initial condition that results in a desired solution at a particular value of `x`. To do that, a rule is created which maps the initial conditions to variables. The PLACE function performs this task, placing the values of `yi` and `zi` into the first elements of the associated lists.

Rules

Status	Rule
* Unsati	place(`y,1) = yi
* Unsati	place(`z,1) = zi

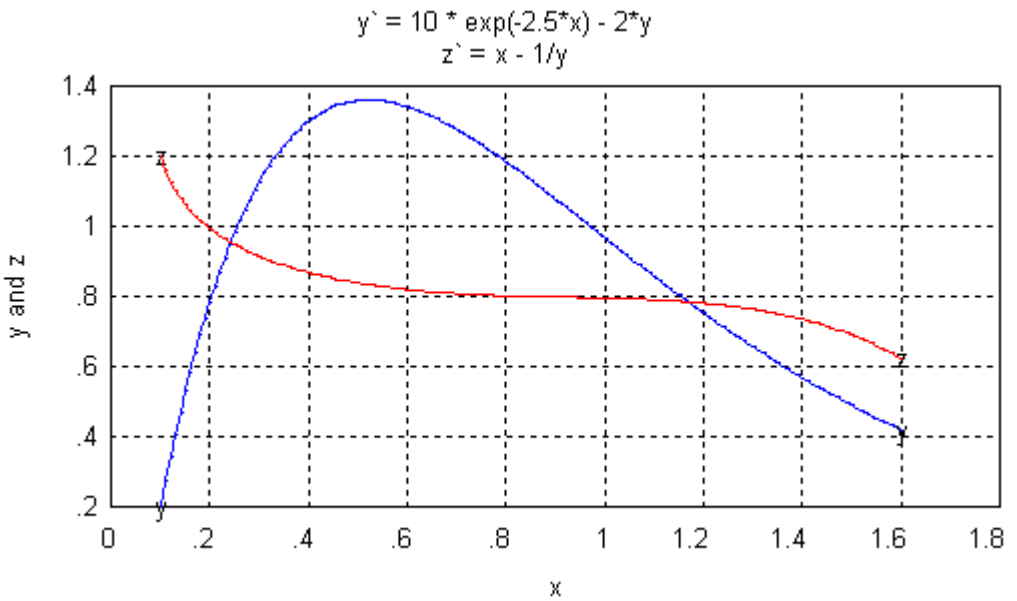
These rules must be inserted *above* the ODE_RK4 function call. Otherwise, they will not be applied until after the integration has been completed using the prior values.

The initial conditions are input on the Variable Sheet.

Variables

Status	Input	Name	Output	Unit	Comment
	1.496927081	x			
		y	.491281936		
		z	.491340014		
	.2	yi			
	1	zi			

Try changing `zi` from 1 to 1.2. The effect can be seen on the plot. The `z` function is raised.

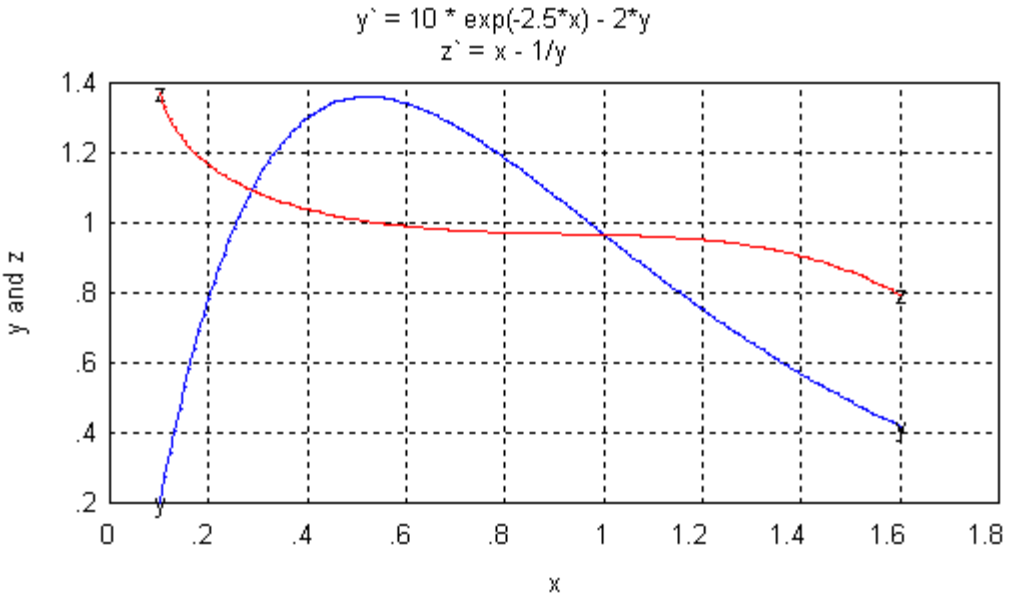


The Optimizer can be used to determine the value of `zi` which forces the `y` and `z` curves to intersect at `x = 1`. The target variable is set as `d` with a target value of 0 again. This time, `zi` is used as the decision variable, with bounds of 1 and 1.5. Here is the solution.

Variables

Status	Input	Name	Output	Unit	Comment
	1	x			
		y	.966057876		
		z	.966057876		
	.2	yi			
	1.370631733	zi			

The plot confirms the solution.

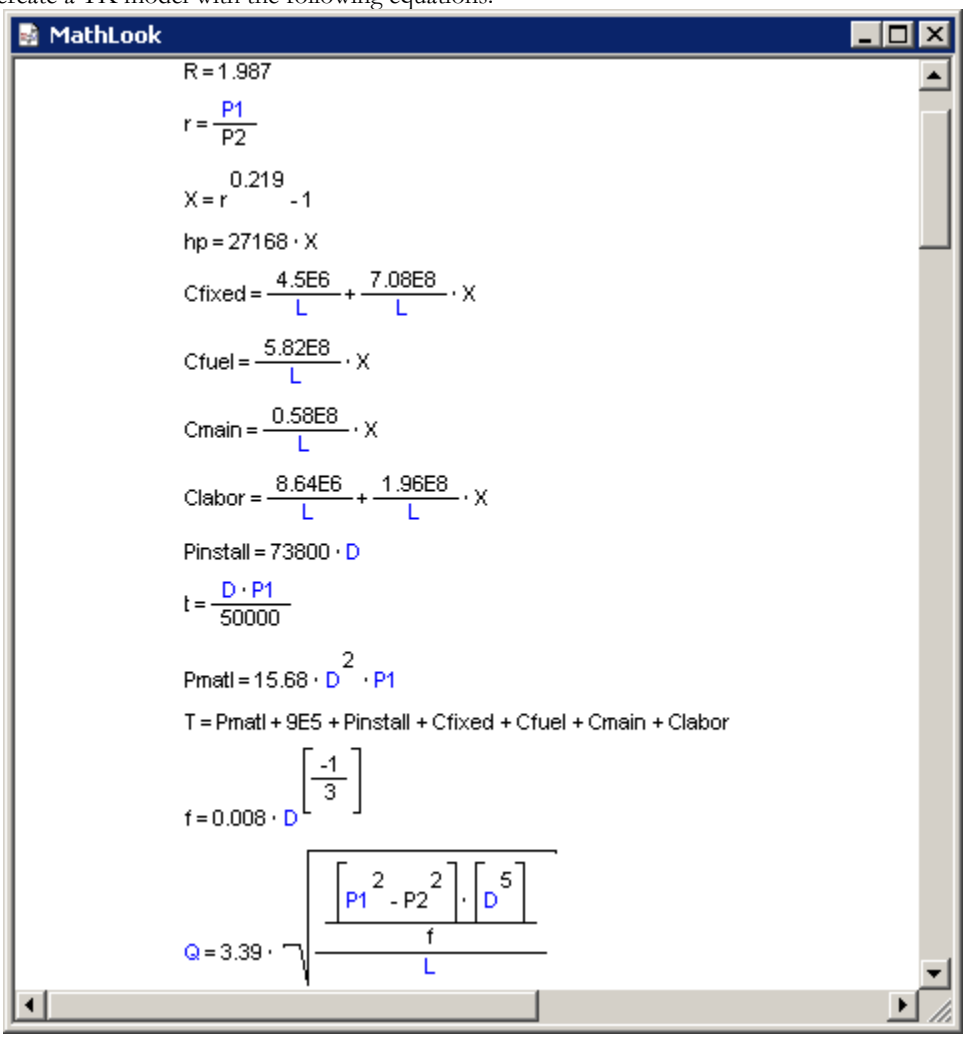


Repeated Optimization, Subject to Constraints

This example is taken from *Engineering Optimization, Methods and Applications* by G.V. Reklaitis, A. Ravindran, and K. M. Ragsdell (John Wiley & Sons, 1983).

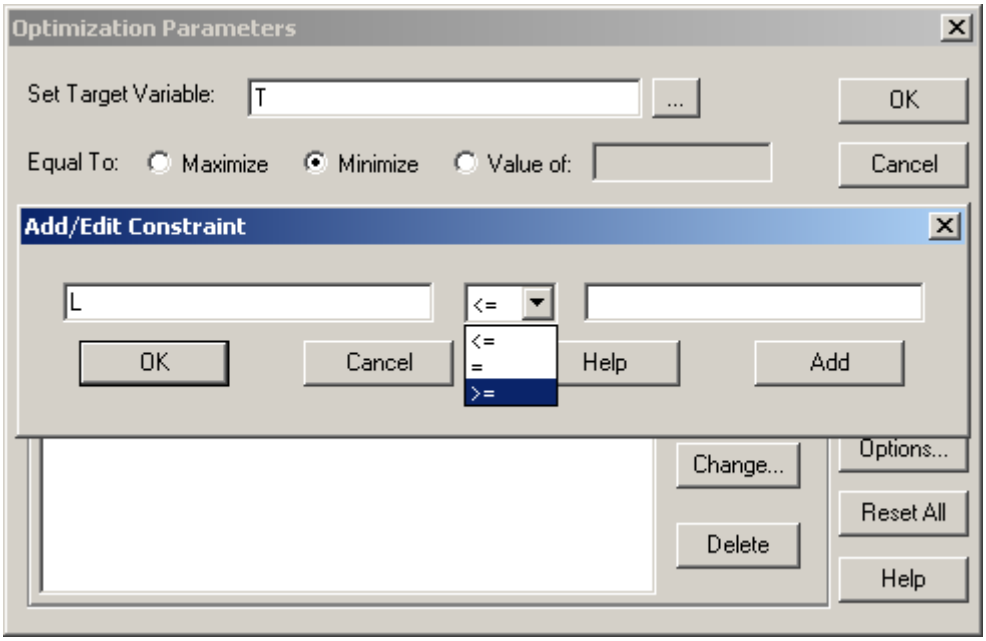
A natural gas pipeline transmission system is required to pump 100×10^{12} cu. ft./day (100 MMSCF/day) of natural gas over a distance of 600 miles. Compressor stations are to be placed at equal distances. The design variables are the pipe diameter `zD`, the compressor discharge pressure `P1`, and the distance between stations, `L`. The optimum design should be such that the total annual cost of amortization and running of the pipeline, `T`, is minimized. If this minimum cost depends on the flow, it should be evaluated for a given set of values.

We create a TK model with the following equations:

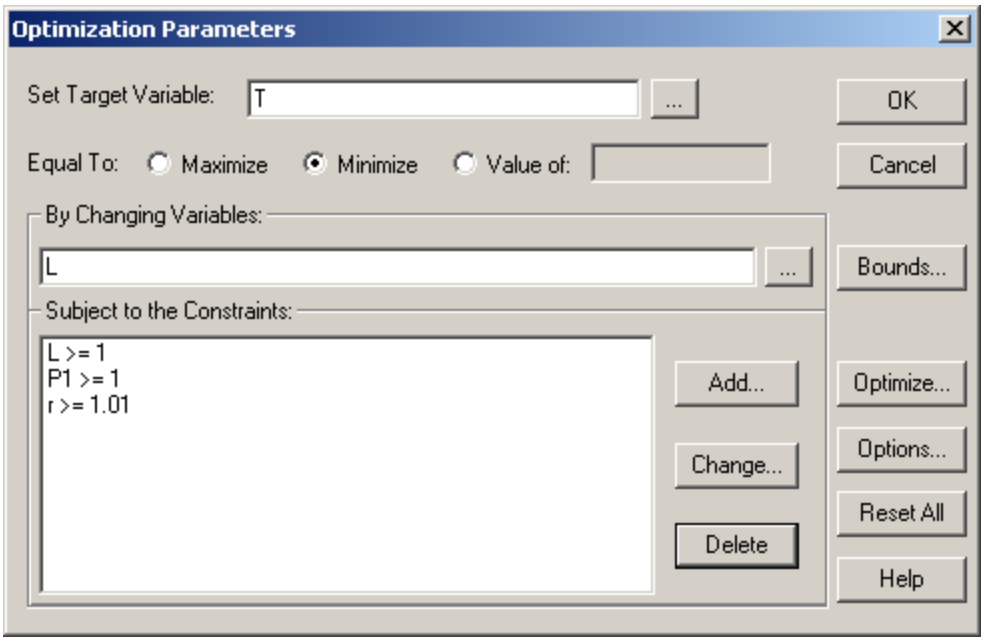

MathLook
R = 1.987
r = P1 / P2
X = r^0.219 - 1
hp = 27168 · X
Cfixed = 4.5E6 / L + 7.08E8 / L · X
Cfuel = 5.82E8 / L · X
Cmain = 0.58E8 / L · X
Clabor = 8.64E6 / L + 1.96E8 / L · X
Pinstall = 73800 · D
t = D · P1 / 50000
Pmatl = 15.68 · D^2 · P1
T = Pmatl + 9E5 + Pinstall + Cfixed + Cfuel + Cmain + Clabor
f = 0.008 · D^[-1/3]
Q = 3.39 · √([P1^2 - P2^2] · [D^5] / f / L)

After testing the model let us seek the ideal input values for a flow of 1,405,790 SCF/hr, proceeding as in the previous examples.

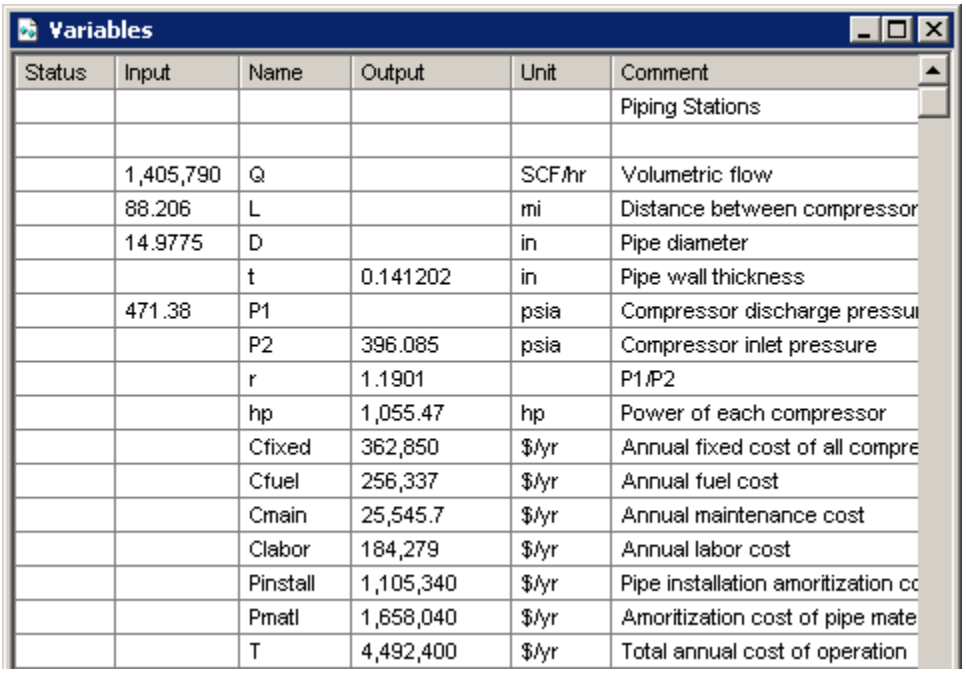
But this time we have constraints: to record them we begin by clicking on the Add… button. This brings up a form for setting those relationships. Enter a series of constraints by using the Add button in the form itself:



The final constraints set is:



Optimizing, we find the best parameters for that flow volume:



Status	Input	Name	Output	Unit	Comment
					Piping Stations
	1,405,790	Q		SCF/hr	Volumetric flow
	88.206	L		mi	Distance between compressor
	14.9775	D		in	Pipe diameter
		t	0.141202	in	Pipe wall thickness
	471.38	P1		psia	Compressor discharge pressu
		P2	396.085	psia	Compressor inlet pressure
		r	1.1901		P1/P2
		hp	1,055.47	hp	Power of each compressor
		Cfixed	362,850	$/yr	Annual fixed cost of all compre
		Cfuel	256,337	$/yr	Annual fuel cost
		Cmain	25,545.7	$/yr	Annual maintenance cost
		Clabor	184,279	$/yr	Annual labor cost
		Pinstall	1,105,340	$/yr	Pipe installation amoritization co
		Pmatl	1,658,040	$/yr	Amoritization cost of pipe mate
		T	4,492,400	$/yr	Total annual cost of operation

In order to store the optimum inputs for flows within a range:

1) We associate lists[2] with the relevant variables, `Q`, `L`, `D`, `P1` and `T`.
2) For handling ease, we collect these lists into a table.
3) We run the Optimizer with the current set of values.
4) We save the results in the relevant lists by using *Command* → *Put Values to Lists…*
5) We repeatedly do the same with the other sets of values.

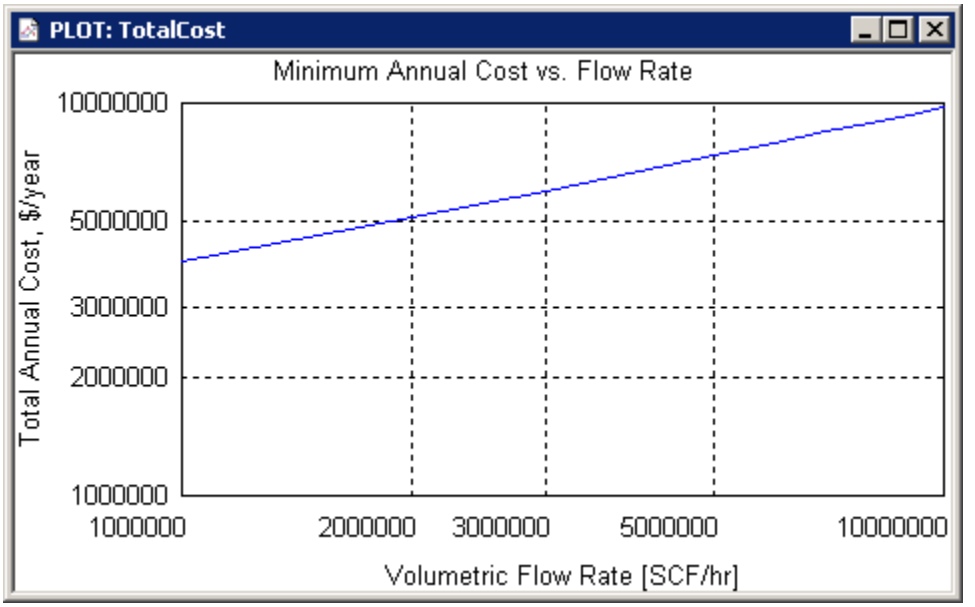
After repeating the process for all values of `Q` we have the following table of results:

INT TABLE: Optimal

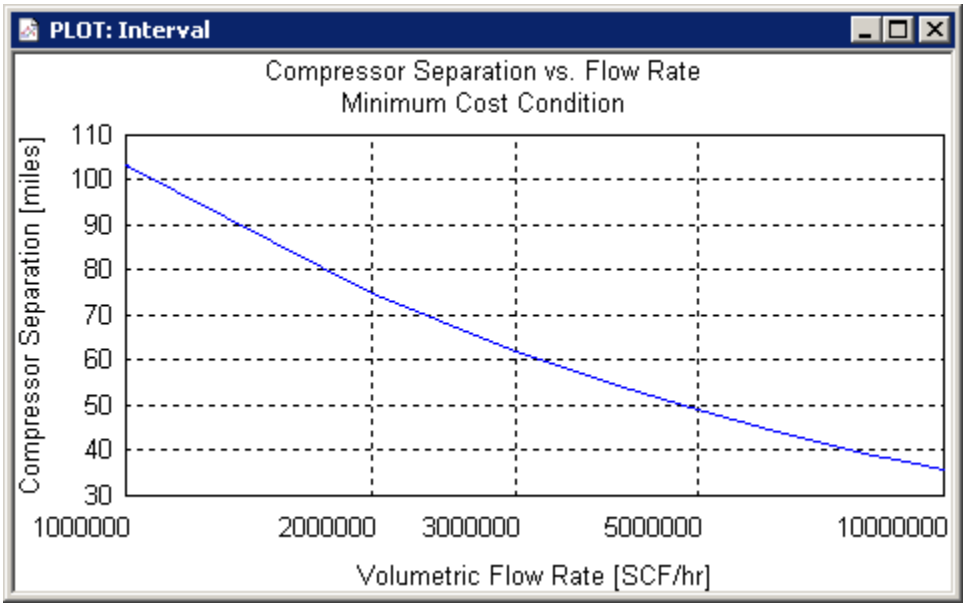
Element	Q	L	D	P1	T
1	10,000,000	35.6569	37.0316	190.74	9,784,880
2	9,000,000	37.4406	35.2849	200.086	9,363,160
3	8,000,000	39.5446	33.4185	211.266	8,915,380
4	7,000,000	42.0815	31.4246	224.65	8,436,300
5	6,000,000	45.1541	29.2608	241.307	7,918,750
6	5,000,000	49.1157	26.8996	262.483	7,352,300
7	4,000,000	54.438	24.2679	290.927	6,720,860
8	3,000,000	62.1831	21.2497	332.286	5,997,100
9	2,000,000	74.9938	17.6244	400.599	5,127,170
10	1,000,000	103.233	12.7982	551.683	3,969,820

It is frequently more informative to present list results in plot form:

[2] Notice that we are NOT going to List Solve (F10)



(using log/log scales)



(or log-linear scales)